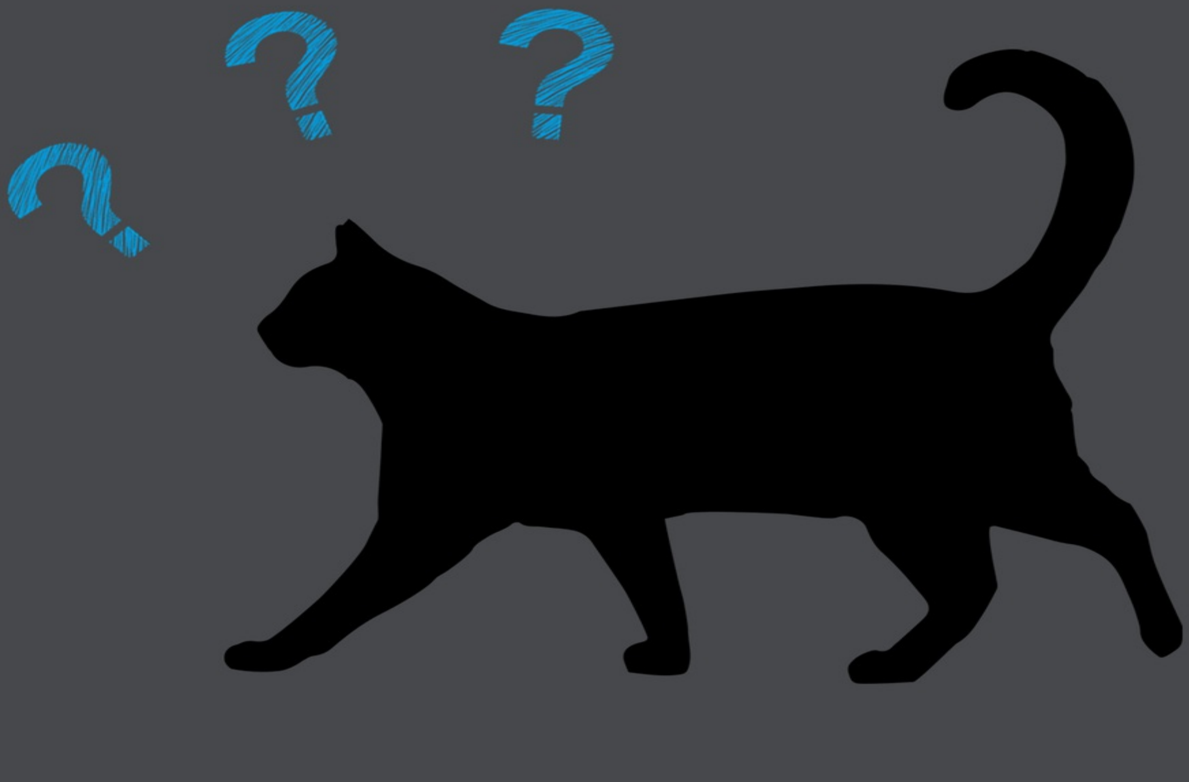
Java Exercise

50 Questions for Beginners

String/Integer/Long/Exception/static initializer/
constructor/File/Files/Zip/Properties/StringBuilder/
LocalDateTime//Random/Array/ArrayList/HashMap/LinkedHashMap/
Functional Interface/Lambda/Stream/Objects/Optional/etc...

Java 8



Java Exercise

50 Questions for Beginners

Introduction

This book is for Java learners especially beginners to intermediate people. This book covers below. Java version is 8.

String/Integer/Long/Exception/static initializer/
constructor/File/Files/Zip/Properties/StringBuilder/
LocalDateTime//Random/Array/ArrayList/HashMap/LinkedHashMap/
Functional Interface/Lambda/Stream/Objects/Optional/etc...

After learning basic concepts and grammar, the next step is to read and understand code. Code of this book includes various methods of classes mentioned above.

There are 50 questions. At the beginning, questions are easy. However, they get more difficult gradually to the end.

The main purpose of this book is to improve code reading comprehension, not grammar theory in detail or something like that.

Please kindly note that there are confusing questions (not recommended in real developing field) for the purpose.

Questions begin from the next page.

Hope you learn and discover something new.

Question 1

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_1 {
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        int c = 3;

        if (a + b == c) {
            System.out.println(method1()[0]);
        }

    }

    private static String[] method1() {
        String[] array = new String[1];
        StringBuilder sb = new StringBuilder();
        sb.append("Be a ");
        sb.append("Java ");
        sb.append("Programmer");
        array[0] = sb.toString();
        return array;
    }
}
```

(A) output Be a Java Programmer
(B) output nothing
(C) compile error
(D) runtime exception

Answer : (A) output “Be a Java Programmer”

Method method1() returns array of String type, and set String from StringBuilder Object to the array[0].

Therefore, this program outputs “Be a Java Programmer” without any error.

Notes :

(1) condition in if statement

You have probably seen simple condition like “a == 1” for your learning, you can also calculate in condition brackets like at line 7 in the question “a + b == c”.

(2) StringBuilder

StringBuilder is frequently used in Java Programming, because it is faster than concatenating by using String object only.

In many cases after that, to covert from StringBuilder object to String object, programmers use method “toString()”.

When you initialize StringBuilder, you can do it in two ways.

First, initialize empty StringBuilder object like at line 15 in the question “StringBuilder sb = new StringBuilder();”.

Second, you can also do it with a String like this “StringBuilder sb = new StringBuilder(“text”);”.

You can simplify your code if you are decided to append first string by using this way.

Question 2

```
public class Question_2_1 {
    private String a = "str";
    private Long b = 1L;
    private int c = 0;
}
```

Select the one statement that describes the result when the program that extends the program above is compiled and run.

```
public class Question_2_2 extends Question_2_1 {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            if (true) {
                if (true) {
                    if (true) {
                        System.out.println(a);
                        System.out.println(b);
                    }
                }
            }
        }
        System.out.println(c);
    }
}
```

(A) output str10
(B) output 0
(C) compile error
(D) runtime exception

Answer : (C) compile error

The access modifiers fields of super class Question_2_1 are all private and non-static. If they are all public and static, compile error doesn’t occur like code below.

```
public class Question_2_1 {
      public static String a = "str";
      public static Long b = 1L;
      public static int c = 0;
}
```

And also, sub class can use fields of Super class, because it extends Question_2_2.

Notes :

(1) condition in if statement

Conditions in if statement requires boolean type, so you can write “true” or “false” in it. If you write true, the inside code will always be executed, and false, the code inside it will never be executed.

It is called “Unreachable Statement”.

One more thing, like this question, writing “true” in if condition is not recommended in real coding situation, because if so, You don’t have to use if statement if the code inside will always be excecuted.

(2) System.out.println()

You have probably seen outputting String with it, but you can output other types of object, like Long, int, and so on. Also, it is better to remember method “System.out.print()” too.

This method outputs something without newline at the end, conversely “System.out.println()”.

Question 3

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_3 {
      public static void main(String[] args) {

            LocalDateTime ldt1 = LocalDateTime.now();
            LocalDateTime ldt2 = ldt1.plusDays(0);

            if (!isLocalDateTime(ldt1)) {
                  return;
            }

            if (ldt2.isAfter(ldt1) || ldt2.isBefore(ldt1)) {
                  System.out.println("ldt1 and ld2 aren't the same");
            }

            System.out.println("ldt1 and ld2 are the same");

      }

      private static boolean isLocalDateTime(LocalDateTime ldt) {

            boolean flg = ldt instanceof LocalDateTime;

            return flg;

      }
}
```

(A) output ldt1 and ld2 are the same
(B) output ldt1 and ld2 aren't the same
(C) output nothing
(D) compile error or runtime exception

Answer : (A) output “ldt1 and ld2 are the same”

In this code, creating two LocalDateTime objects. The second object, “ldt2” is made by “ldt1” with method “plusDays();”.

However, They are same as LocalDateTime object, because the parameter is “0”;

For the above reason, the statement at line 12 is not be executed, and the program outputs “ldt1 and ld2 are the same”.

Notes :

(1) LocalDateTime

This class is added from Java 8. Default format is below.

“yyyy-MM-ddTHH:mm:ss.SSS”

You can use this class when you want objects with timestamp, not only date or time.

Method “now()” creates a LocalDateTime object with datetime when you run the program.

And methods “isAfter()” and “isBefore” compare two LocalDateTime objects wether the one is after or before the other.

(2) return

There are two ways to use this statement. You can return the result of a method, or stop the program on where you write it.

(3) instanceof

This operator checks if the parameter is an object of given Class. At line 21 in this question, it checks if “ldt” is LocalDateTime Class object.

In this case, “flg” is true;

Question 4

Select the one statement as code replacement for “// some code” in code below to output “JAVA” when this program is compiled and run.

```
public class Question_4 {

    public static void main(String[] args) {

        LinkedHashMap<String, String> map =
                new LinkedHashMap<String, String>();

        map.put("1", "J");
        map.put("2", "A");
        map.put("3", "V");
        map.put("4", "A");

        // some code

        if (flg) {
            map.entrySet().stream().forEach(e -> {
                System.out.print(e.getKey());
            });
        } else {
            map.entrySet().stream().forEach(e -> {
                System.out.print(e.getValue());
            });
        }

    }
}
```

(A) `boolean flg = true;`
(B) `boolean flg = false;`
(C) `Boolean flg = null;`
(D) `Object flg = null;`

Answer : (B) boolean flg = false;

In this code, creating “map” object and putting keys and values. The goal is to output the values inside “map” object in adding order.

In condition of if statement at line 15, If “flg” is true, the program will output all keys, and false, all values. So the answer is (B).

Notes :

(1) LinkedHashMap

When you start learning Java, you learn HashMap first in most cases. In this case however, you have to output all values in adding order.

Hashmap doesn’t guarantee the order, so this is why this code uses LinkedHashMap instead.

(2) choice (C) and (D)

About choice (C) : If you write “Boolean flg = null;” in your code, compile error doesn’t occur, but if you write “boolean flg = null;” in your code, it occurs.

This is the difference between primitives and objects. Primitives are values themselves. And objects provide reference, and null means there is no reference. Therefore, Objects are called “nullable”.

About choice (D) : Object class wraps all classes, so compile error doesn’t occur if you write it in your code.

(3) forEach

This is a method in StreamAPI that added from Java 8. It does something with all elements inside the stream.

It is better for beginners to think it as an alternative way to write for loop of collection. Strictly, they are not the same, but it is enough at first.

Question 5

```
public class Question_5_1 {
      public Integer number;
      public Integer getNumber() { return number;  }
      public void setNumber(Integer number) { this.number = number; }
}
```

Select the one statement that describes the result when the program that uses the program above is compiled and run.

```
public class Question_5_2 {
      public static void main(String[] args) {
            Question_5_1 instance1 = new Question_5_1();
            Question_5_1 instance2 = new Question_5_1();
            instance1.setNumber(0);
            instance2.setNumber(0);

            if (instance1.getNumber() == instance2.getNumber()) {
                  System.out.print("true");
                  if (instance1.equals(instance2)) {
                        System.out.print(" " + "true");
                  } else {
                        System.out.print(" " + "false");
                  }
            }
      }
}
```

(A) output true true
(B) output true false
(C) output nothing
(D) compile error or runtime exception

Answer : (B) output true false

In this code at line 8, comparing set number of two fields of Question_5_1 instances.

It returns true, so the program outputs something anyways and this is why choice (C) is wrong. And the condition at line 10 returns false, so choice (B) is correct.

Notes :

(1) not starting a new line inside curved brackets

Starting a new line inside curved brackets is frequently seen in Java programming.

Surely, by doing that, you can improve readability of your code if there are multiple statements, but there is only one statement inside curved brackets, not starting a new line is good way to write code.

Comparison :

<not starting a new line>

```
{ // some code here; }
```

<starting a new line>

```
{
      // some code here;
}
```

(2) equals method of Object Class

At line 10, comparing if the two instances are the same “instance” .

As you can see at line 5 and 6, value of fields are same, but they are different instances. This is why the condition returns false;

Question 6

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_6 {

      public static void main(String[] args) {

            while (!false) {
                  try {
                        System.out.print("A");
                        Integer.parseInt("Hello World");
                  } catch (Exception e) {
                        System.out.print("B");
                        break;
                  } finally {
                        System.out.print("C");
                  }
            }

            System.out.print("D");
      }

}
```

(A) output AD
(B) output ABD
(C) output ABCD
(D) output ABDC

Answer : (C) output ABCD

At line 5, !false means the same as true, so the loop starts.

An exception occurs at line 8 because of the argument isn’t number, so the statement in both catch and finally block will be executed.

Notes :

(1) Integer.parseInt()

This is a static method of Integer class. It converts the String argument to int value, and is frequently seen in Java programming.

When the arguments is not number (as value, not as type), NumberFormatException occurs. This is one of the most famous exception as NullPointerException.

(2) Processing order of break statement and finally block

As you can see at line 11, the program will get out of while loop as a result.

Finally block will be executed as priority, and then get out of while loop next.

(3) catch block

In this code, you only see one catch block, but you can handle each exception by writing multiple catch blocks like below.

You should write code like this if you want different result depends on each exception.

```
} catch (NullPointerException e) {
      // some code here;
} catch (NumberFormatException e) {
      // some code here;
}
```

Question 7

Select the one statement that describes the result when this program is compiled and run.

```
----------------------------------------------------------------------

public class Question_7 {

      public static void main(String[] args) {

            String str1 = "Hello";
            String space = " ";
            Question_7 question_7 = new Question_7();

            question_7.method1(str1 + space);
            System.out.println(str1);

      }

      private void method1(String str1) {

            String str2 = "World";
            str1 += str2;

      }

}

----------------------------------------------------------------------
```

(A) output Hello
(B) output Hello World
(C) compile error
(D) runtime exception

Answer : (A) output Hello

At line 9, giving parameter “Hello ” to method1.

In method 1, str1 changes from “Hello “ to “Hello World”, but this code outputs “Hello”.

Let’s see it in detail from below.

Notes :

(1) value

The parameter that is given method1 at line 9 is value, so the variable str1 in main method is different from str1 ini method1. It’s just copy.

The variable str1 in method1 changes as you can see at line 17, but it doesn’t affect str1 in main method. Thus, this program outputs “Hello” not “Hello World”.

(2) static and non-static method

Main method is static, but method1 is not. If you rewrite code at line 9 like “method1(str1 + space);”, compile error occurs.

This is called static reference. If both methods are static, compile error doesn’t occur.

(3) operator “+=”

This operator means the same as below.

```
String str1 = “ABC”;
String str2 = “DEF”;

// same
str1 = str1 + str2;
str1 += str2;
```

Question 8

When this program is compiled, compile error occurs. Select the one statement that describes certain line where it does.

```
 1  public class Question_8 {
 2
 3        public static void main(String[] args) {
 4
 5              final List<String> list1 = new ArrayList<>();
 6
 7              list1.add("This ");
 8              list1.add("list ");
 9              list1.add("is ");
10              list1.add("final.");
11
12              List<String> list2 = new ArrayList<>();
13
14              System.out.println(list2.size());
15
16              list1 = list2;
17
18              System.out.println(" done.");
19
20        }
21
22  }
```

(A) line 7
(B) line 14
(C) line 16
(D) line 18

Answer : (C) line 16

At line 5, list 1 is initialized as final ArrayList. If you assign another list to final ArrayList, compile error occurs.

Notes :

(1) ArrayList

<final ArrayList>

As mentioned above, final ArrayList can’t be assigned after it is initialized.

You might think why compile error doesn’t occur at line 7. It is good point of view. About final ArrayList, it is permitted to add new elements conversely reassigning.

<method size()>

You might also think why compile error doesn’t occur at 14, it is good point of view, too. Surely list2 is initialized, and that’s it.

No elements are added after that, but using method size() is no problem. It simply returns 0.

<way to initialize>

In this code at line 5 and 7 to 10, used 5 lines to create ArrayList with four elements.

It doesn’t causes any error or run time exception.

However, there is another way to initialize it without using fewer lines like below and it is better to remember.

```
final List<String> list1 = Arrays.asList("This ", "list ", "is ",
"final.");
```

Question 9

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_9 {

    public static void main(String[] args) {

        HashMap<Integer, Integer> map = new HashMap<>();

        map.put(1, 2);
        map.put(3, 4);

        map.clear();

        if (map.isEmpty()) {
            map.put(1, 1);
            map.put(1, 2);
        }

        System.out.println(map.get(1) > map.size());
    }

}
```

(A) output true
(B) output false
(C) compile error
(D) runtime exception

Answer : (A) output true

At line 17, map.get(1) returns 2 and map.size() returns 1, so this program outputs true.

Let’s take a look at reasons why it outputs so.

Notes :

(1) HashMap

<method clear()>

This method is really simple. If you use it, all entry sets in a map will be deleted. The map will be empty.

In this code, putting elements at line 7 and 8, but at line 10, the map will be empty.

<method isEmpty()>

This method is really simple one, too.

It checks whether a map has entry sets or not, so at line 12, the condition returns true.

<duplicate key>

At line 13 and 14, putting elements with duplicate key 1.

In this case, the entry set put earlier will be overwritten because of the next key of entry set is duplicate.

This is why “map.get(1) > map.size()” returns true at line 17. The overwritten value is 2, and size is 1.

Question 10

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_10 {
      public static void main(String[] args) {

            Integer a = method1("1");

            if (a == 0 && a != null) {
                  System.out.println(a);
            }

      }

      private static Integer method1(String str) {

            if (Integer.parseInt(str) > 0) {
                  return null;
            } else {
                  return 100;
            }

      }
}
```

(A) output null
(B) output 100
(C) compile error
(D) runtime exception

Answer : (D) runtime exception

At line 12 method1 returns null, because the argument is greater than 0.

After that, runtime exception occurs at line 6, so the answer is (D).

Notes :

(1) NullPointerException

This exception is one of the most famous exception in Java Programming.

In most cases, Java programmer encounter this exception at least one time.

<if condition in this code>

In this code at line 6, comparing null and 0 with operator “==”, this is why the exception occurs.

To evaluate both conditions “a == 0” and “a != null”, reverse the order of them. By doing so, you can do null check first, and then if it is null, it returns false and runtime exception doesn’t occur.

<happens normally>

The exception is frequently seen in many times.

For instance, someone who is in a hurry for relaseing application and write code without null check.

After release of application, it stopped due to NullPointerException because of the code without null check.

This doesn’t happen if there is enough time to do tests, but sometimes no time. So, it is not a strange case and happens normally.

Question 11

Select the one statement that describes the result when this program is compiled and run.

```
------------------------------------------------------------------------
```

```
public class Question_11 {
      public static void main(String[] args) {

            String fileName = "file1.csv";
            String fileNameNoExtension =
                        fileName.substring(0, fileName.indexOf("."));

            if (method1(fileName)) {
                  System.out.println(fileNameNoExtension +
                              " is csv file");
            } else {
                  System.out.println(fileNameNoExtension +
                              " isn't csv file");
            }

      }

      public static boolean method1(String value) {
            return value.endsWith(".csv");
      }
}
```

```
------------------------------------------------------------------------
```

(A) output file1.csv is csv file
(B) output file1.csv isn’t csv file
(C) output file1 is csv file
(D) output file1 isn’t csv file

Answer : (C) output file1 is csv file

What this program does is to check if file1 is csv file, and output the result with filename without extension.

You can learn about several methods of String class from this.

Notes :

(1) String

<method indexOf()>

This method returns the number of occurrence of the argument in String, so “fileName.indexOf(".")” at line 6 returns 5.

It counts number with the same way as array elements, file1.csv consists of 8 characters, and the number of “.” occurrence is 5.

<method substring()>

This method returns string cut off with specified range. Below is how to understand it.

```
 f i l e 1 . c s v
| | | | | | | | | |
0 1 2 3 4 5 6 7 8 9
```

If you get string cut off with range 0 to 5 above, you can get fileName without extension like in the code of this Question at line 5.

<method endsWith()>

This method is easy to understand compared to above two. As you can guess by just reading method name, it checks if String ends with the argument.

If so, it returns true and not, it returns false.

Question 12

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_12 {

      public static void main(String[] args) {

            String str = "123";

            if (method1(str)) {
                  Object a = Integer.parseInt(str);
                  System.out.println((String) a);
            }

      }

      public static boolean method1(String value) {
            try {
                  Integer.parseInt(value);
                  return true;
            } catch (NumberFormatException e) {
                  return false;
            }
      }

}
```

(A) output 123
(B) output nothing
(C) compile error
(D) runtime exception

Answer : (D) runtime exception

The if condition at line 7 returns true because value of str is number.

However, runtime exception occurs at line 9 because object a is Integer type. From below, let's take a look at it in detail.

Notes :

(1) ClassCastException

This exception occurs when casting an object to type that is not inheritance relation. String and Integer at line 8 and 9 is a good example of this.

<at line 8>

At line 8, indicating Object type when initializetion of object a, so compiler doesn't realize type of value assigned until the program is run.

Below is two way of writing line 8 in this code.

1. Object a = Integer.parseInt(str);
2. Integer a = Integer.parseInt(str);

If you rewrite code at line 8 with way 2, compile error occurs at line 9, because of indicating Integer type when initializing object a tells compiler what type of object a.

(2) method1

At line 14, name of method1 is method1. However, in real coding situation, name of this type of method starts with "is" usually. For instance, "isString", "isNumber", "isFile", "isDir(Dir means directory)".

You should name methods like above if it checks the argument whether something and returns boolean value.

Question 13

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_13 {

    public static void main(String[] args) {
        String array[] = { "a", "b", "c" };

        for (int i = 0; i < array.length; i++) {
            if (isArrayEmpty(array)) {
                return;
            }
            array[i] = array[i].toUpperCase();
        }

        System.out.println(array[array.length]);
    }

    public static boolean isArrayEmpty(String[] array) {
        return array == null || array.length == 0;
    }

}
```

(A) output C
(B) output null
(C) compile error
(D) runtime exception

Answer : (D) runtime exception

This code seems it outputs the last element of array that is converted to uppercase.

The way of thinking is correct, but it is wrong to use returned value of array.length when outputting the last elements of an array.

Notes :

(1) toUpperCase()

This method is at line 10. It is really simple and easy to understand. It converts lower case string to upper-case.

(2) array

<operate all elements>

At line 6 to 11, this for loop is frequenlty seen to operate all elements in an array, so all elements of array changes to upper-case like below.

before : { "a", "b", "c" }
after : { "A", "B", "C" }

<get the last element>

In this code, array length returns 3 because there are three elements in it. However, to get the last elements of it, need to write array[2].

At line 13, array[array.length] is equals to write array[3], so runtime exception occurs.

(3) ArrayIndexOutOfBoundsException

This is the exception that occurs when this program is run.

It is better to remember the exception because this is frequently seen exception in real coding situation.

Question 14

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_14 {
      public static void main(String[] args) {

            Random random = new Random();
            int a = 0;

            for (int i = 0; i < 10000; i++) {

                  a = random.nextInt(10);

                  if (a > 9) {
                        System.out.println("abc");
                        return;
                  }

            }

            System.out.println("ABC");
      }
}
```

(A) output abc
(B) output ABC
(C) compile error
(D) runtime exception

Answer : (B) output ABC

At line 7 to 16, checking if field a is greater than 9 10000 times.

However, there is no possibility that a is greater than 9, because method nextInt() at line 9 returns random number between 0 to 9.

As a result, this program outputs ABC.

Notes :

(1) Random

<method nextInt()>

As mentioned above, this method returns random number between 0 to the argument.

<Rock, Paper, Scissors Game>

Below is another example of using this method, and this program is simple Rock, Paper, Scissors Game.

```
public static void main(String[] args) {

      String[] array = {
                  "Rock",
                  "Paper",
                  "Scissors"
      };

      System.out.println(array[new Random().nextInt(3)]);

}
```

Create array with Strings Rock, Paper, Scissors. And when outputting array member, giving random number by method nextInt() to index of array. Thus, this program will output Rock, Paper, or Scissors randomly when is compiled and run.

Question 15

There are three text files in directory(C:/Users/user/Desktop/).

<directory (C:/Users/user/Desktop/)>

```
-------------------------------------------------------------------------
file1.txt
file2.txt
file3.txt
-------------------------------------------------------------------------
```

Select the one statement that describes the result when this program is compiled and run.

```
-------------------------------------------------------------------------

public class Question_15 {

      public static void main(String[] args) {

            File dir = new File("C:/Users/user/Desktop/");

            Arrays.stream(dir.listFiles()).forEach(file -> {
                  System.out.print(file.getName() + " ");
            });

      }

}

-------------------------------------------------------------------------
```

(A) output file1.txtfile2.txtfile3.txt
(B) output null
(C) compile error
(D) runtime exception

Answer : (A) output file1.txtfile2.txtfile3.txt

What this program does is below.

1. get all files in directory (C:/Users/user/Desktop/)
2. output names of all files

Notes :

(1) File

<create file object>

File class is good choice to operate file objects.

As you can see at line 5, you can also use File class if you want to create directory object by giving file path to the constructor of File class.

<method listFiles()>

This method is frequently seen. At line 7, it returns all files in a directory as array of File type.

<method getName()>

At line 8, you can see the method. It returns only file name. If you output “file” not “file.getName()”, it outputs each file objects with full path.

(2) Arrays

At line 7, method forEach() is the same as one in Question 4. It’s in StreamAPI, but array of File type doesn’t have method stream(), so what this program between line 7 to 9 does is below.

1. create stream of array of File type
2. use method forEach() of the stream
3. output each element

The parameter file before arrow (->) is single element in the stream.

Question 16

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_16 {
      public static void main(String[] args) {

            String str1 = "a";
            String str2 = "b";
            String str3 = "c";

            List<String> list = Arrays.asList("a", "b", "c");

            list.stream().forEach(e -> {
                  switch (e) {
                  case str1:
                        System.out.println("switch");
                        break;
                  case str2:
                        break;
                  case str3:
                        break;
                  }
            });

      }
}
```

(A) output switch
(B) output nothing
(C) compile error
(D) runtime exception

Answer : **(C) compile error**

At line 4 to 6, there are all variables.

However, case clauses in switch statement needs final variable, so compile error occurs at line 12, 15, 17.

Notes :

(1) switch statement

Switch statement is good choice if there are many conditions compared to if statement.

<three conditions using if-else statement>

```
if (condition1) {
      return;
} else if (condition2) {
      return;
} else {
      return;
}
```

<three conditions using switch statement>

```
1  switch ("") {
2  case condition1 : break;
3  case condition2 : break;
4  default : break;
5  }
```

At line 4 of above code “default” means the same as else of if statement.

Another point of which to use if statement or swtich statement is whether conditions are simple or complicated.

Switch statement needs final variables and looks simple, but if statement can evaluate complicated conditions easier than swtich statement.

Question 17

Below is content of file (C:/Users/user/Desktop/abc.properties).

```
key=value

```

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_17 {
      public static void main(String[] args) throws Exception {
            Properties property = new Properties();
            File file = new File("C:/Users/user/Desktop/abc.properties");
            FileInputStream stream = new FileInputStream(file);
            String str = "key";

            property.load(stream);
            property.clear();
            property.setProperty(str, "java");

            System.out.println(property.getProperty(str));

            stream.close();
      }
}
```

(A) output value
(B) output java
(C) output null
(D) compile error or runtime exception

Answer : **(B) output java**

(1) What this program does is below

1. create FileInputStream from properties file.
2. load it and make it empty
3. set key and value and output it
4. close the stream (to avoid memory leak)

Notes :

(1) properties file

This kind of file ends with an extension ".properties".

This is just simple key-value content, and used for storing configurations of Java Applications.

(2) Properties

This class is for operating properties file that mentioned above.

<method load()>

At line 8, this method is load properties file. You should simply load it by giving FileInputStream to this method.

<method clear()>

After loading properties file, object property has one entry set "key" and "value" in this case. If you use this method, object property has no entryset. It will be empty.

<method setProperty() and getProperty()>

Both method is really simple to understand. You can set key-value content, or get it with a key. If there is no entry set in property object, method getProperty() returns null.

Question 18

Select the one statement that describes "//condition here" to make method shuffleText() of below code never return the same value as the given argument.

```
--------------------------------------------------------------------------

public class Question_18 {
    public static void main(String[] args) {
        System.out.println(shuffleText("java"));
    }

    public static String shuffleText(String text) {
        while (true) {
            List<String> list = new ArrayList<>();
            StringBuilder sb = new StringBuilder();

            for (char character : text.toCharArray()) {
                list.add(String.valueOf(character));
            }

            Collections.shuffle(list);
            list.forEach(e -> sb.append(e));

            if (//condition here) {
                return sb.toString();
            }
        }
    }
}

--------------------------------------------------------------------------
```

(A) sb.toString().equals(text)
(B) !sb.toString().equals(text)
(C) true
(D) false

Answer : **(B) !sb.toString().equals(text)**

What method shuffleText() of this program does is below

1. loop until return shuffled text except same value as the argument
2. create list and add each character of the argument
3. shuffle the list add to StringBuilder
4. return String value from the StringBuilder object

Notes :

(1) toCharArray()

This method is frequently seen in operating each character of String like at line 11 to 13.

Remember to convert to String after operating like line at 12 with using method String.valueOf();

(2) Collections.shuffle()

This method is really useful. If you want to shuffle the elements of a list, just write this and give a list as an argument and that’s it.

(3) answer of this question

The condition of choice (B) “!sb.toString().equals(text)” checks if the shuffled String is different from the arguemnt.

If so, return it. (because this code is suppose not to return same value as the given argument.)

However, with the condition of choice (A) “sb.toString().equals(text)” at line 18, this code will return only same value as the given argument.

In this code, it always returns “java”.

Question 19

The method Character.isLetter() returns true if the argument is alphabet. Select the one statement that describes “**// code here**” to make method isAlphabet() of below code return true if the argument of this method is alphabet.

```java
public class Question_19 {

      public static void main(String[] args) {
            List<String> list = Arrays.asList("abc", "def", "123");
            for (String str : list) {
                  if (!isAlphabet(str)) {
                        System.out.println(str);
                  }
            }
      }

      public static boolean isAlphabet(String text) {
            int cnt = 0;
            for (char character : text.toCharArray()) {
                  if (!Character.isLetter(character)) {
                        cnt++;
                  }
            }
            Return // code here;
      }

}
```

(A) cnt < 1
(B) cnt > 1
(C) cnt < 2
(D) cnt > 2

Answer : **(A) cnt < 1**

What method isAlphabet() of this program does is below

1. create charArray from the argument
2. operate all character of it
3. if it is not alphabet, increment cnt
4. return if cnt is less than 1 (never incremented)

Notes :

(1) When to use this code in coding situation

<method isAlphabet()>

The method isAlphabet() returns boolean value as a result of checking the argument.

This type of method is used for validation.

For instance, sent message from parameters input from users, file names inside directory, a certain column in database, and so on.

<at line 5 to 8>

In real coding situation, this code can be used for outputting or storing all non-alphabet Strings for error message.

If the elements of the list at line 4 are file names, users can know all bad file names for certain application.

However, if the validation returns error message right after only one lements is non-alphabet, users can't know all the elements.

So, this style of coding is very useful when to output error messages.

Question 20

Below is inside of zip file (C:/Users/user/Desktop/file.zip).

```
----------------------------------------------------------------------
file1.txt
file2.txt
file3.txt
----------------------------------------------------------------------
```

Select the one statement that describes the result when this program is compiled and run.

```
----------------------------------------------------------------------

public class Question_20 {
      public static void main(String[] args) throws Exception {

            File file = new File("C:/Users/user/Desktop/file.zip ");
            ZipInputStream stream =
                        new ZipInputStream(new FileInputStream(file));
            ZipEntry entry = stream.getNextEntry();

            while (entry != null) {
                  System.out.print(entry.getName());
                  stream.closeEntry();
                  entry = stream.getNextEntry();
            }

            stream.close();
      }
}

----------------------------------------------------------------------
```

(A) output file1.txtfile2.txtfile3.txt
(B) output nothing
(C) compile error
(D) runtime exception

Answer : **(A) output file1.txtfile2.txtfile3.txt**

What this program does is below

1. create ZipInputStream from the zip file
2. loop if element of them is not null and output file name

Notes :

(1) zip file handling

There code uses only standard Java library, not any external library, and is basic code of handling zip file in Java.

Below is processing order in detail.

1. After creating ZipInputStream, create ZipEntryObject. This object is one element (file) in zip file.

2. In while loop, check if the ZipEntryObject is not null, outputting fileName, and close the ZipEntryObject each loop.

And assigning next ZipEntryObject to variable entry like incrementing number at the last of loop.

3. Close both ZipEntryObject and ZipInputStream like at line 11 and 15.

(2) nested stream

You might realize why this code doesn’t close FileInputStream at line 5. That is a good point of view.

In Java, you don’t have to close each stream if it is nested. Only you have to do is to close wrap stream.

In this code, it is ZipInputStream. After you close it, nested stream FileInputStream closes, too.

Question 21

Select the one statement that describes the result when this program is compiled and run.

```java
public class Question_21 {

	public static void main(String[] args) {

		String str1 = "A B C, ABC ";
		String[] array = str1.split(",");

		for (String str : array) {
			method1(str.toLowerCase().trim());
		}

		for (String str : array) {
			method1(str);
		}

	}

	private static void method1(String value) {
		System.out.print(value);
	}

}
```

(A) output a b cabcA B C ABC
(B) output a b cabca b c abc
(C) output abcabcABCABC
(D) output abcabcabcabc

Answer : (A) output a b cabcA B C ABC

What this program does is below.

1. create array with elements “A B C”, and “ ABC ”.
2. output each of them by two ways, changing lower case and removing spece at the beginning and the end, and as it is.

Notes :

(1) method toLowerCase()

This method simply converts upper-case character to lower-case in String.

(2) method trim()

This method removes space at the beginning and the end, so spaces except them remains.

This is why choice (C) and (D) are wrong.

(3) method chaining

At line 9, there are multiple methods on str toLowerCase() and trim().

This is called “method chaining”.

(4) why choice (B) is wrong

String is immutable.

At line 9, it seems changing value of str, but it is wrong.

The value of str never changes, so it is need to assign to another variable if you want to keep using changed value.

If String is mutable, the answer is (B).

Question 22

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_22 {

      public static void main(String[] args) {

            String str1 = null;
            String str2 = str1 == null ? null : "";

            try {
                  System.out.print(str2.trim());
            } catch (Exception e) {

                  System.out.print(str2 == null ? 0 : 1);
                  System.exit(1);

            } finally {
                  System.out.print(str2 == null ? 0 : 1);
            }

      }

}
```

(A) output 00
(B) output 11
(C) output 0
(D) output 1

Answer : (C) output 0

The point of this question is if finally block will be executed or not.

Notes :

(1) way to stop application

<return>

If there is “return” statement in code, you can stop the method. However, it means just returning to caller, so the application continues to run.

If you rewrite code at line 13 from “System.exit(1)” to “return”, the answer is choice (A).

<method System.exit()>

On the contrary, this method stops the application.

This is why the statement in finally block is not be executed, so the answer is choice (C).

Also, the argument of this method is 0 and 1 usually, and 0 means program finished with no error, and 1 means program finished with error.

(2) ternary operator

At line 6, 12 and 16, there are ternary operators. Below is how to think this code.

1. before question mark, it is condition that returns boolean value
2. after that, which one of “0 : 1” will be returned
3. if the condition before question mark is true, 0 will be returned, or if the condition before question mark is false, 1 will be returned.

This is useful because you don’t need to write null check code, for instance.

Question 23

Select the one statement that describes the result when this program is compiled and run.

```java
public class Question_23 {
    public static void main(String[] args) {
        List<String> list1 = Arrays.asList("a", "b");
        List<String> list2 = Arrays.asList("c", "d");
        List<String> list3 = Arrays.asList("e", "f");
        List<List<String>> lists = Arrays.asList(list1, list2, list3);

        List<String> concatenatedList = concatList(lists);
        concatenatedList.forEach(e -> System.out.print(e));
    }

    public static List<String> concatList(List<List<String>> lists) {
        List<String> list = new ArrayList<>();

        for (List<String> e : lists) {
            for (String str : e) {
                list.add(str);
            }
        }

        return list;
    }
}
```

(A) output abcdef
(B) output fedcba
(C) output nothing
(D) compile error or runtime exception

Answer : (A) output abcdef

The point of this code is to concatenate multiple lists.

Compared to other questions, you might think this difficult, so let’s take a look at how this code works from below.

Notes :

(1) how this code works

1. At line 6, create a list that stores lists.

This might be difficult to understand if you have only seen list like “List<String>” or “List<Integer>”, and so on.

2. First, in method concatList(), create a list that will be returned. Second, with two for loops, it concatenate all lists.

Let’s see first loop at line 15. The object e is each list that added at line 6 to lists, so the second for loop is need to add all String elements in it.

And in second loop, str is each object of each list at line 3 to 5.

They are “a”,”b”,”c”,”d”,”e”,”f”; By adding these all elements to the list at line 13, concatenating is completed.

(2) with streamAPI

Below is alternative code to write at line 15 to 19.

```
lists.forEach(x -> {
      x.forEach(y -> {
            list.add(y);
      });
});
```

Question 24

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_24 {

      public static void main(String[] args) {
            List<Integer> list1 = Arrays.asList(1, 2, 3, 4, 5, 6);
            List<Integer> list2 = list1.stream()
                        .filter(e -> e % 2 == 0)
                        .collect(Collectors.toList());

            System.out.println(add(list2));
      }

      public static int add(List<Integer> list) {

            int number = 0;

            for (int e : list) {
                  number += e;
            }

            return number;

      }

}
```

(A) output 10
(B) output 20
(C) output 12
(D) output 24

Answer : (C) output 12

What this code does is below.

1. create first list with elements 1, 2, 3, 4, 5, 6
2. create second list with elements with even number of first list
3. add all of them and output the result of calculation

Notes :

(1) StreamAPI

<method filter()>

This method returns boolean value. You can write condition to filter elements in the stream like at line 7 in this code.

Below is alternative code at line 7. You can write like this, too.

```
.filter(e -> {
      return e % 2 == 0;
})
```

Below is easy code to get odd or even number. You can use them as they are.

odd : number % 2 != 0
even : number % 2 == 0

<method collect()>

This method is frequently seen in Java code that using streamAPI, in case that creating a new list from another list.

The code at line 8 “.collect(Collectors.toList());” returns list.

It usually comes at the end of stream operation.

Question 25

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_25 {
      public static void main(String[] args) {
            String[] array = { "a", "b", "c", "5" };
            String str = null;

            Arrays.stream(array).forEach(e -> {
                  if (e.matches("^[0-9]*$")) {
                        str = e;
                  }
            });

            System.out.println(padWithZeroLeft(str, 5));
      }

      public static String padWithZeroLeft(String value, int digits) {
            String zero = "0";
            StringBuilder sb = new StringBuilder();

            for (int i = 1; i < digits; i++) {
                  sb.append(zero);
            }

            return sb.append(value).toString();
      }
}
```

(A) output 00005
(B) output 000005
(C) compile error
(D) runtime exception

Answer : (C) compile error

At line 6 to 10, it is lambda expression. In lambda expression, you can’t change value of variable in it. It should be final.

If code at line 6 to 10 is below, the answer is (A).

```
for (String e : array) {
      if (e.matches("^[0-9]*$")) {
            str = e;
      }
}
```

Notes :

(1) regular expression

At line 7, you can see “^[0-9]*$”. This means 0 to 9.

(2) method matches()

With the above regular expression, this checks if string is number or not.

It returns boolean value.

(3) **method padWithZeroLeft()**

Zero padding is frequently seen String operation in real coding situation.

There are external library to do it, and of course you can create it by yourself like at 15 to 24 in this code.

To fix this method better(means fewer bugs when real use), the first argument “value” should be checked if null or not.

And adding exception handling, and make variable “zero” final because it isn’t supposed to be changed.

Question 26

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_26 {

      public static void main(String[] args) {
            int a = 1;
            int b = Arrays.asList(100, 200).size();
            int c = new Long(3L).intValue();

            if (a + b == c && (c - a) % 2 == 0) {
                  System.out.println(method1()[0]);
            }
      }

      private static String[] method1() {
            String[] array = new String[1];
            StringBuilder sb = new StringBuilder();

            sb.append("1 ".trim());
            sb.append(sb.toString().toUpperCase());
            array[0] = sb.toString();

            return array;
      }

}
```

(A) output 1
(B) output 11
(C) compile error
(D) runtime exception

Answer : (B) output 11

There are a few points that seems a bit complicated. Let’s take a look at it one by one.

Notes :

(1) line 5

Arrays.asList(100, 200).size() means the size of the ArrayList that consists of two elements 100 and 200, so it returns 2.

(2) line 6

Firstly, Long type is created by using “new Long(3L)”. However, if you assign it to variable c causes compile error, because they are different type each other.

The method intValue() helps situation like this, it returns int value of te Long object.

(3) line 8

The second part of the condition ((c - a) % 2 == 0) looks a bit complicated.

Let’s divide it into two. The variable c is 3, and a is 1 in this code, so “(c - a)” is 2, and it’s even number, so this condition returns true.

(4) line 17

The method trim() is seen in former questions, but this case seems wrong.

In this code, it is used on double quoted String, not variable. However, it is no problem, so error doesn’t occur.

(5) line 18

When operating StringBuilder, you can use the object itself like this line.

Question 27

```
public class Question_27_1 {
      private static String str1 = "A";
      public static String concatStr(String str1, String str2) {
            return new StringBuilder()
                       .append(str1)
                       .append(str2)
                       .toString();
      }
      public static String getStr1() {
            return str1;
      }
}
```

Select the one statement that describes the result when the program that extends the program above is compiled and run.

```
public class Question_27_2 extends Question_27_1 {
      public static void main(String[] args) {
            getStr1().toLowerCase();
            Question_27_2 instance = new Question_27_2();
            instance.method1();
      }

      private void method1() {
            System.out.println(concatStr(getStr1(), getStr1()));
      }
}
```

(A) output aa
(B) output AA
(C) compile error
(D) runtime exception

Answer : (B) output AA

The points of this code is below.

1. accessing field and method of super class
2. way of writing return statement (at line 4 of super class)

Notes :

(1) accessing field and method of super class

The access modifier of super class field str1 is private.

However, there is method getStr1(), too. The access modifier of the method is public, so any error doesn’t occur if you use this method to get the field instead of accesing it directly.

This type of method is called “getter method” or simply “getter”.

On the contrary there is another type of method, called “setter method” or simply “setter”. The access modifier is usually public like as getter, and this method is to set value to private field of a certain class. Below is how it looks like (if this question’s code).

```
public static void setStr1(String str1) {
      Question_27_1.str1 = str1;
}
```

(2) way of writing return statement (at line 4 of super class)

There are many cases return value that is already initialized before return statement, and that is frequently seen coding style.

However, you don’t have to create value before return statement, you can do it with return statement.

Just write it right after “return” like this code at line 4.

Question 28

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_28 {
      public static void main(String[] args) {
            LocalDateTime ldt1 =
                        LocalDateTime.of(2020, 01, 01, 12, 34, 56);
            if (!isLocalDateTime(ldt1)) {
                  return;
            }
            if (!isLocalDateTime(ldt1)) {
                  System.exit(1);
            }

            System.out.println(ldt1.getYear());
      }

      public static boolean isLocalDateTime(LocalDateTime ldt) {
            Predicate<LocalDateTime> predicate =
            new Predicate<LocalDateTime>() {
                  public boolean test(LocalDateTime e) {
                        return e instanceof LocalDateTime;
                  }
            };
            return predicate.test(ldt);
      }
}
```

(A) output 2020
(B) output nothing
(C) compile error
(D) runtime exception

Answer : (A) output 2020

What this program does is below.

1. create LocalDateTime object
2. check if it is LocalDateTime type
3. output year of it

Notes :

(1) LocalDateTime

<method of()>

To create LocalDateTime object, use method now() or method of().

Method now() returns LocalDateTime object at the timestamp of when the application is run.

And, you can specify values of timestamp by using method of().

<method get Year()>

This method returns year of LocalDateTime object. There are more methods like this, getMonth(), getHour() and so on.

(2) Predicate

This is one of Functional Interfaces from Java 8. This is to return boolean value.

In this code at line 16 to 21, it isn’t lambda expression.

You can rewrite it with lambda expression like below.

```
Predicate<LocalDateTime> predicate = e -> e instanceof LocalDateTime;
```

And to use above code, write same code at line 22.

Question 29

Select the one statement that describes the result when this program is compiled and run.

```
------------------------------------------------------------------------

public class Question_29 {

      public static void main(String[] args) {

            HashMap<Object, String> map = new HashMap<Object, String>();

            map.put(1L, "A");
            map.put(null, "B");
            map.put("String", null);

            Question_29 instance = new Question_29();
            instance.method1(map.get("String"));

      }

      public void method1(Object object) {

            Optional<Object> a = Optional.ofNullable(object);
            System.out.println(a.orElse("output"));

      }

}

------------------------------------------------------------------------
```

(A) output null
(B) output output
(C) compile error
(D) runtime exception

Answer : (B) output output

What this program does is below.

1. create Hashmap with three entry sets
2. if object of value from key “String” of the map is not null, output it.
3. if object of value from key “String” of the map is null, output “output”.

Notes :

(1) Object type as map key

Object type is nullable, so if you indicate it as key of map, you can set null to it.

Also, Object type wraps all types, so 1L and “String” can be set.

(2) Optional

Optional type is for objects that might be null. It wraps the object.

<method ofNullable()>

The argument of the method1 might be null, so the method is used. If the argument might not be null, you can use method of() instead.

<method orElse()>

This method is for nullable value.

If you use if statement, you have to write code like below instead of code at line 19.

```
if (object != null) {
      System.out.println(object);
} else {
      System.out.println("output");
}
```

Question 30

```java
public class Question_30_1 {
      private Long a;
      public Long getA() { return a; }
      public void setA(Long a) { this.a = a; }
}
```

Select the one statement that describes the result when the program that uses the program above is compiled and run.

```java
public class Question_30_2 {
      public static void main(String[] args) {
            Question_30_1 a = new Question_30_1();
            Question_30_1 b = new Question_30_1();

            a.setA((long) 1);
            b.setA(Long.valueOf(2));

            List<Question_30_1> list = Arrays.asList(a, b);
            method1(list);
      }

      public static void method1(List<Question_30_1> list) {
            list.forEach(new Consumer<Question_30_1>() {
                  public void accept(Question_30_1 object) {
                        System.out.print(object.getA());
                  }
            });
      }
}
```

(A) output 12
(B) output 1212
(C) compile error
(D) runtime exception

Answer : (A) output 12

What this program does is below.

1. create two instances of Class with one field
2. set value each value of them
3. create list with the instances
4. output each field of element of the list

Notes :

(1) convert to Long

There are three easy ways to convert object to long

1. (long) object
2. Long.valueOf(object)
3. new Long(object)

No.2 and 3 are Long, but 1 is long. It converts to Long automatically by Java compiler.

(2) Consumer

This is one of Functional Interfaces from Java 8. The method of this doesn’t return value, type is void.

In this code at line 14 to 18, it isn’t lambda expression.

You can rewrite it with lambda expression like below :

```
list.forEach(e -> System.out.print(e.getA()));
```

or below :

```
list.forEach(e -> {
      System.out.print(e.getA());
});
```

Question 31

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_31 {
      public static void main(String[] args) {

            Long a = 100L;
            Integer b = 100;

            if (Integer.compare(a.intValue(), b) == 0
                        && b instanceof Integer) {

                  try {

                        String str = null;
                        Objects.requireNonNull(str, "ABC");

                  } catch (NullPointerException e) {

                        System.out.println(e.getMessage());

                  }

            }
      }
}
```

(A) output nothing
(B) output ABC
(C) compile error
(D) runtime exception

Answer : (B) output ABC

The point of this code is below.

1. what method Integer.compare does
2. what method Objects.requireNonNull does

Notes :

(1) method Integer.compare()

This method uses two arguments and compare whether the first argument is equeal to, greater, or less than the second argument.

It returns int value. Return values are below.

1. The arguments are equal : 0
2. The first is greater than second : 1
3. The first is less than second : -1

So, return value of this code at line 7 is true.

(2) method Objects.requireNonNull()

This method at line 13 uses two arguments.

It checks whether the first argument is null or not, and if it is null, throws NullPointerException.

The second argument is used for message of the exception, so, this code outputs ABC.

However, there is one important point.

If you rewrite code at line 17 to “System.out.println(“Java”);” for example, this code outputs “Java”.

The second argument only affects the message field of NullPointerException object, but above code doesn’t use it.

Question 32

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_32 {

      public static void main(String[] args) {

            List<Integer> list1 = Arrays.asList(1, 2, 3);
            List<Integer> list2 = Arrays.asList(4, 5, 6);

            List<List<Integer>> list3 = Arrays.asList(list1, list2);

            method1(list3).forEach(System.out::print);

      }

      private static List<Object> method1(List<List<Integer>> list) {

            return list.stream()
                        .flatMap(e -> e.stream())
                        .sorted(Comparator.reverseOrder())
                        .collect(Collectors.toList());

      }

}
```

(A) output 123456
(B) output 654321
(C) compile error
(D) runtime exception

Answer : (B) output 654321

What this program is below.

1. create two lists to store Integer elements
2. create one list to store them
3. merge two list in the list to one list to store Integer with reverse order
4. output them all

Notes :

(1) method flatMap()

This method changes multiple collections into one stream. It makes it easy to concat list, for example.

In this code, there are two lists in one list at line 8 like below.

Inside of list3 :

1. [1, 2, 3]
2. [4, 5, 6]

After using method flatmap() and method sorted() :

[6, 5, 4, 3, 2, 1]

(2) method reference

At line 10 “method1(list3).forEach(System.out::print);” is using method reference. You can shorten your code with this.

If you rewrite it to lambda, it looks like below.

```
method1(list3).forEach(e -> System.out.print(e));
```

Question 33

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_33 {

    public static void main(String[] args) {

        String str = "JAVA";

        try {
            throw new NumberFormatException();
        } catch (NullPointerException e) {
            return;
        } catch (Exception e) {

            Predicate<Long> a = Predicate.isEqual(2L);
            if (a.test(10L)) {
                str.replace("JAVA", "ABC");
            } else {
                str.replace("JAVA", "");
            }

        }

        System.out.println(str.length());

    }

}
```

(A) output 0
(B) output 3
(C) output 4
(D) compile error or runtime exception

Answer : (C) output 4

This code does is below.

1. initialize str
2. throw NumberFormatException explicitly
3. second catch block, replacing str
4. output length of str

Notes :

(1) throw exception explicitly

It seems a bit strange to throw exception explicitly, but it doesn’t cause any error.

The exception throwed at line 8 is NumberFormatException, so the first catch block isn’t executed.

However, the second catch block catches all exceptions, because Excepion class wraps all of it.

(2) Predicate

What code at line 13 and 14 does simple. It compares 2L and 10L and returns boolean value.

This is way to use method isEqual() in Predicate Interface.

(3) String

String is immutable. If code at line 14 returns true and at line 15 is “str = str.replace("JAVA", "ABC");”, the answer is (B).

If code at line 14 returns false and at line 17 is “str = str.replace("JAVA", "");”, the answer is (A).

Question 34

Select the one statement that describes the result when this program is compiled and run.

```java
public class Question_34 {
    public static void main(String[] args) {

        Question_34 a = new Question_34();
        ABC b = a.new ABC(false);

        System.out.println(b.str1);

    }

    class ABC {
        private String str1;

        public ABC(boolean flg) {
            Predicate<Boolean> a = new Predicate<Boolean>() {
                public boolean test(Boolean flg) {
                    return !flg;
                }
            };

            str1 = a.test(flg) ? "a" : "b";
        }
    }
}
```

(A) output a
(B) output b
(C) compile error
(D) runtime exception

Answer : (A)

What this code does below

1. create inner class instance b with false
2. output str of the instance

Notes :

(1) Inner Class

<inner class and constructor>

In Java, you can create inner class inside of outer class like at line 11.

Inner class ABC at line 11 has one field, and the constructor is at line 14 to 22.

The constructor uses one boolean type argument, so if you create instance of this class without any argument or wrong parameter, it causes compile error.

And the method test() of Predicate interface at line 15 uses Boolean type argument and return reverse value of it, so the parameter that will be assigned to str1 is a.

<create instance>

Then, how to use inner class is a bit complicated.

If you write code like below in method main(), it causes compile error.

```
ABC instance = new ABC();
```

This means when creating instance of inner class, you have to create instance of outer class first.

After that, you can create inner class instance with code like at line 5.

Question 35

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_35 {

      private static final String str1;
      private static final String str2;

      static {
            if (Question_35.class.getSimpleName().length() < 0) {
                  str1 = "";
                  str2 = "";
            } else {
                  str1 = "A";
                  str2 = "B";
            }
      }

      public static void main(String[] args) {
            Optional<String> a = Optional.ofNullable(str1);
            a.ifPresent(new Consumer<String>() {
                  @Override
                  public void accept(String t) {
                        System.out.println(str2);
                  }
            });
      }

}
```

(A) output A
(B) output B
(C) output nothing
(D) compile error or runtime exception

Answer : (B) output B

What this program is below.

1. initialize two constants in static initializer
2. output str2

Notes :

(1) static initializer

Static initializer is static code that will is executed only once when class is loaded.

For example, when you want to initialize constant with if statement, static initializer is useful.

As you can see at line 3 and 4, access modifiers are both final, but you can assign value in static initializer.

(2) method getSimpleName()

This method is in Class class.

It returns class name simply as the method name indicates.
So, return value of “Question_35.class.getSimpleName()” is Question_35.

(3) method ifPresent()

This is useful method in Optional class. The code at line 18 to 23 means the same as below.

```
if (a != null) {
      System.out.println(Question_35.class.getSimpleName());
}
```

(4) override annotation

This method isn’t required, but recommended to avoid overriding mistakes.

Question 36

This is inside of file.txt (C:/Users/user/Desktop/file.txt)

```
a,b,c,d,e
A,B,C,D,E
1,2,3,4,5

```

Select the one statement that describes the result when this program is compiled and run.

```java
public class Question_36 {
      public static void main(String[] args) throws IOException {
            Path path = Paths.get("C:/Users/user/Desktop/file.txt");

            List<String> list1 = Files.readAllLines(path);
            List<String> list2 = list1.stream()
                        .map(e -> e.split(","))
                        .flatMap(e -> Arrays.stream(e))
                        .map(e -> e.toLowerCase())
                        .distinct()
                        .collect(Collectors.toList());

            list2.forEach(e -> {
                  if (e.matches("^[0-9]*$")) {
                        System.out.print("0");
                  }
            });

      }
}
```

(A) output 00000
(B) output 0000000000
(C) compile error
(D) runtime exception

Answer : (B) output B

What this program does is below.

1. read all lines from file.txt
2. create list with elements [a, b, c, d, e, 1, 2, 3, 4 ,5]
3. output 0 five times (loop the list and check if element is numeric)

Notes :

(1) method readAllLines()

This method in Files class is really useful.

If you read all lines in text file, you have to write just only one line code like at line 3 in this code.

(2) at line 6 to 11

If you are not used to stearmAPI, this looks a little bit complicated. What these lines do is below.

1. all lines in file change to three String arrays
2. they change to one stream
3. change all upper-case character to lower-case in it
4. delete duplicate value in it
5. return list

As mentioned above, method map() and flatMap() are used to change something.

(3) StreamAPI Operations

They are two operations in streamAPI. They are intermediate operation and terminal operation.

In this code, at line 7 to 10 are intermediate operations.

And, at line 11 is terminal operation.

Question 37

Select the one statement that describes the result when this program is compiled and run.

```java
public class Question_37 {
    public static void main(String[] args) {
        Function<String, Boolean> a =
                new Function<String, Boolean>() {
            @Override
            public Boolean apply(String t) {
                return null;
            }
        };

        if (null == null) {
            method1(a);
        }

    }

    public static void method1(Function<String, Boolean> a) {
        try {
            Optional<Boolean> b = Optional.of(a.apply("123"));
        } catch (NullPointerException e) {
            System.exit(1);
        } finally {
            System.out.println("321");
        }
    }
}
```

(A) output 321
(B) output nothing
(C) compile error
(D) runtime exception

Answer : (B) output nothing

This code does is below

1. create function interface
2. catch NullPointerException
3. program stops

Notes :

(1) Function

This is one of Funcional Interfaces. The default method of this is apply(), so you can see override annotation at line 5.

To understand easily, think it as normal method in Java. At line 3 and 4, There are two generic arguments.

It means the function uses String argument, and returns Boolean value.

So, it does is same as code below.

```
<access modifier> Boolean method1(String str) {
      Return null;
}
```

(2) at line 11 to 13

It seems like compile error, but it works fine. It returns true. However, it is not recommended in real code situation.

(3) method of()

This method of Optional class is to wrap non-nullable value. It throws NullPointerException if the value is null, so after executing at line 19, the exception will be caught.

And, finally block doesn’t executed if application ends with method System.exit(), so the answer is (B).

Question 38

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_38 {

      public static void main(String args[]) throws Exception {

            int min = 1, a = 0, b = 0;
            Integer[] array = { 2, 5, 1, 3, 4 };

            Arrays.parallelSort(array);
            a = array[0];

            Thread.sleep(1000);

            Arrays.sort(array);
            b = array[0];

            if (min == a && a == b) {
                  Integer[] array2 = Arrays.copyOf(array, 6);
                  System.out.print(Arrays.toString(array2));
            } else {
                  return;
            }

      }

}
```

(A) output [1, 2, 3, 4, 5, null]
(B) output nothing
(C) compile error
(D) runtime exception

Answer : (A) output [1, 2, 3, 4, 5, null]

What this program does is below.

1. initialize three int variables
2. initialize array of Integer type with five elements
3. sort it twice
4. create copy of it
5. output it

Notes :

(1) initialize multiple variable

It seems like compile error at line 5, but you can initialize multiple variables with one line. This helps code looks simple.

You don’t have to use new line for each variable.

(2) method sort() and parallelSort()

These method are used for sorting array. What they do is the same actually, but processing speed is different.

The method parallelSort() is multi-threaded, so it is faster than method sort() if array is large.

(3) method sleep()

This method is in Thread Class. It simply pauses application with specified time. In this code at line 11, the argument 1000 is equal to 1 second.

(4) method copyOf()

This method is in Arrays class. In this code at line 17, giving 6 as second argument. Actually it is greater than length of array, but it doesn’t throw exception.

As answer (A) says, it creates copy of array with null.

Question 39

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_39 {

    public static void main(String args[]) {

        List<Integer> list1 = new ArrayList<>();
        List<Integer> list2 = new ArrayList<>();

        list1.addAll(Arrays.asList(new Integer[] { 1, 2, 3, 4, 5 }));
        list2.addAll(Arrays.asList(new Integer[] { 1, 2, 3 }));

        list2.addAll(list1.subList(3, 5));

        if (list1.equals(list2)) {

            list2.clear();
            list2.sort(Comparator.naturalOrder());
            list2.sort(Comparator.reverseOrder());

            System.out.print(list2.get(0));

        } else {
            return;
        }

    }

}
```

(A) output []
(B) output nothing
(C) compile error
(D) runtime exception

Answer : (D) runtime exception

What this program does is below.

1. initialize two list with different elements
2. add elements to second list and elements are both same
3. try to output first element of empty list

Notes :

(1) method addAll()

This method is used for add elements to list by using argument of list.

(2) method subList()

To understand this method easily, remembering method substring() in String class is good way. It takes specified range of elements of list like below.

```
 1 2 3 4 5
| | | | | |
0 1 2 3 4 5
```

At line 11 in this code, arguments are 3 and 5, so 4 and 5 of elements of list1 is added to list2 from the end.

(3) method equals() in List

This method returns true if type, size, elements, and orders are equal.

(3) method clear()

It makes list empty. At line 16 and 17, trying to sort element list, but it doesn’t throw exception.

(4) IndexOutOfBoundsException

This exception occurs when you try to get elements greater range of size of list. In this code, it is at line 19.

Question 40

Select the one statement that describes the result when this program is compiled and run.

```java
public class Question_40 {
    class ABC {
        private String str;
    }

    public static void main(String args[]) {
        Question_40 a = new Question_40();
        ABC b = a.new ABC();
        ABC c = a.new ABC();
        ABC d = a.new ABC();

        b.str = "A";
        c.str = "B";
        d.str = "C";

        List<ABC> list = Arrays.asList(b, c);
        list.add(d);

        for (ABC e : list) {
            if (e instanceof ABC) {
                System.out.print(e.str);
            } else {
                System.exit(1);
            }
        }
    }
}
```

(A) output ABC
(B) output nothing
(C) compile error
(D) runtime exception

Answer : (D) runtime exception

What this program does is below.

1. create three instances of inner class ABC
2. set value of each field of them
3. use them for elements of list
4. try to add last element and exception occurs

Notes :

(1) initializing ArrayList

Initializing ArrayList with “Arrays.asList()” is frequently seen code. However there is one important point to do this.

There are two types of list like below.

1. mutable list
2. immutable list

Mutable list means that you can’t add element after initialized it.

If you create list with “Arrays.asList()”, it’s a mutable list, so you can’t add elements.

On the contrary, if you initialize list with “new ArrayList<>()”, it’s a immutable list.

So you can add elements without problem.

Incidentally, finally list is mutable list, too. Finally in this case, means that you can’t reassign another list to it.

However, adding elements is no problem.

Question 41

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_41 {

      private  static  Function<StringBuilder,  String>  function  =  new
Function<StringBuilder, String>() {
            @Override
            public String apply(StringBuilder sb) {
                  return sb.delete(0, sb.length() - 1)
                              .append("bc123")
                              .toString();
            }
      };

      public static void main(String[] args) {
            String str = function.apply(new StringBuilder("java"));

            List<String> list = new ArrayList<>();
            for (char e : str.toCharArray()) {
                  if (Character.isLetter(e)) {
                        list.add(String.valueOf(e));
                  }
            }

            list.forEach(System.out::print);
      }
}
```

(A) output abc
(B) output bc
(C) compile error
(D) runtime exception

Answer : (A) output abc

What this program does is below.

1. create String object that value is abc123
2. create list with only alphabet values of it
3. output each element

Notes :

(1) Function

This is one of Funcional Interfaces.

It means the function uses StringBuilder argument, and returns String value. So, it does is same as code below.

```
private static String method1(StringBuilder sb) {
      return sb.delete(0, sb.length() - 1)
                  .append("bc123")
                  .toString();
}
```

This function is executed at line 14.

(2) method delete() in StringBuilder

To understand this method easily, remembering method substring() in String class is good way.
It deletes specified range of elements of StringBuilder like below.

```
 j a v a
| | | | |
0 1 2 3 4
```

So, the return value of this method at line 7 is “a” and append “bc123” and convert it to String as a result.

Question 42

Select the one statement that describes the result when this program is compiled and run.

```java
public class Question_42 {
      class ABC {
            private HashMap<Integer, String> map;

            public ABC(int a) {
                  for (int i = 0; i < a; i++) {
                        map.put(i, "java" + (i + 1));
                  }
            }
      }

      public static void main(String args[]) {
            Question_42 a = new Question_42();
            ABC b = a.new ABC(5);

            List<Object> list = b.map.entrySet().stream()
                        .map(e -> e.getValue())
                        .collect(Collectors.toList());

            list.forEach(new Consumer<Object>() {
                  public void accept(Object object) {
                        System.out.print(object);
                  }
            });
      }
}
```

(A) output java0java1java2java3java4
(B) output java1java2java3java4java5
(C) compile error
(D) runtime exception

Answer : (D) runtime exception

What this program does below.

1. create instance of class Question
2. create instance of class ABC
3. throws NullPointerException in constructor of class ABC

Notes :

(1) constructor of class ABC

<why NullPointerException occurs>

1. use argument of int type as how many time it loops
2. try to put elements of map field
3. throws NullPointerException at line 7

At line 3, map is created. However, it is null at this point.

So, when trying to put elements with method put() throws NullPointerException.

<avoid causing NullPointerException>

This program runs without exception when you rewrite code at line 3 as below.

```
private HashMap<Integer, String> map = new HashMap<>();
```

If so, after executing code in constructor, entryset of map looks below.

```
0:java1, 1:java2, 2:java3, 3:java4, 4:java5
```

This is why answer is (B).

If the method getValue() is getKey(), the Answer is output 01234.

Question 43

Select the one statement that describes the result when this program is compiled and run.

```java
public class Question_43 {

    public static void main(String args[]) {

        Boolean flg = null;
        Integer[] array = { 10, 20, 30 };

        int a = array[0], b = array[1];
        flg = Integer.compare(a + b, array[2]) != 0 ? null : !false;

        Optional<Boolean> c = Optional.ofNullable(flg);

        c.ifPresent(x -> {
            for (Integer y : array) {
                y = 0;
            }
        });

        Arrays.stream(array).forEach(System.out::print);

    }

}
```

(A) output 102030
(B) output 000
(C) compile error
(D) runtime exception

Answer : (A) output 102030

What this program does is below.

1. create array of Integer type with elements [10, 20, 30]
2. check if 10 + 20 is equal to 30
3. loop the array
4. output each element

Notes :

(1) Integer.compare()

This method uses two arguments and compare whether the first argument is equeal to, greater, or less than the second argument. It returns int value. Return values are below.

1. The arguments are equal : 0
2. The first is greater than second : 1
3. The first is less than second : -1

So, return value of this code at line 9 is true, and value of flg is true.

(2) standard for loop, enhanced for loop, forEach of StreamAPI

There are three ways to loop array, but there is only one way that affects the root array like below.

1. standard for loop (affects root array)
2. enhanced for loop (doesn’t affect root array)
3. forEach of StreamAPI (doesn’t affect root array)

This is why the answer is (A).

If you use forEach of StreamAPI at line 14 to 16, the answer is also (A).

If you use standard for loop at line 14 to 16, the answer is (B).

Question 44

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_44 {

      public static void main(String args[]) {

            List<String> list1 = Arrays.asList("a", "b", "c");
            List<String> list2 = list1.stream()
                        .filter(e -> !e.matches("^[0-9]*$"))
                        .sorted(Comparator.reverseOrder())
                        .collect(Collectors.toList());

            List<String> list3 = null;

            try {
                  if (list1.equals(list2)) {
                        System.exit(1);
                  } else {
                        Objects.requireNonNull(list3, "list3 is null");
                  }
            } catch (Exception e) {
                  System.out.println("Exception occurred");
            }

      }

}
```

(A) output list3 is null
(B) output Exception occurred
(C) compile error
(D) runtime exception

Answer : (B) output Exception occurred

What this program does is below.

1. create list1 with elements ["a", "b", "c"]
2. create list2 with elements of list1 except number with reverse order
3. create list3 as null
4. compare if both list1 and list2 are equal
5. throws NullPointerException
6. output Exception occurred

Notes :

(1) regular expression

At line 7, you can see “^[0-9]*$”. This means 0 to 9. However, there is no number in elements ["a", "b", "c"], so this code doesn’t affect actually.

(2) method equals() in List

This method returns true if type, size, elements, and orders are equal.

As you can see at line 8, with method sorted(), order of list2 changes to reverse order, so return value is false at line 14.

(3) method Objects.requireNonNull()

This method at line 17 uses two arguments.

It checks whether the first argument is null or not, and if it is null, throws NullPointerException. The second argument is used for message of the exception.

However, the second argument only affects the message field of NullPointerException object, but this code at line 20 doesn’t use it.

This is why the answer is not (A).

Question 45

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_45 {

      private static Supplier<Question_45> a =
                        new Supplier<Question_45>() {
            @Override
            public Question_45 get() {
                  return new Question_45();
            }
      };

      public static void main(String args[]) {
            Question_45 b = a.get();
            Question_45 c = a.get();

            HashMap<Integer, String> map = new HashMap<Integer, String>();
            map.put(1, "a");
            map.put(1, "b");
            if (b.equals(c)) {
                  return;
            } else {
                  Optional<String> d = Optional.of(map.get(2));
                  System.out.print(d.isPresent());
            }
      }

}
```

(A) output null
(B) output false
(C) compile error
(D) runtime exception

Answer : (D) runtime exception

What this program does is below.

1. create Supplier that returns instance of Class Question
2. create map and put one element (because it’s duplicate)
3. try to get value that doesn’t exit in it (with key 2)
4. throws NullPointerException after try to wrap it with method of()

Notes :

(1) Supplier

This is one of Functional Interfaces from Java 8. This is to return new instance of Class.

The method get() at line 6 is executed at line 12 and 13.

(2) equals method of Object Class

At line 18, comparing if the two instances are the same “instance” .

As you can see at line 12 and 13, value of fields are same, but they are different instances. This is why the condition returns false;

(3) method of()

This method of Optional class is to wrap non-nullable value.

It throws NullPointerException if the value is null, so after executing at line 21, the exception occurs.

This is why answer is (D).

If code at line 21 is with method ofNullable() instead of method of(), the answer is (B).

Method ofNullable() can wrap null value without throwing exception.

Question 46

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_46 {
      public static void main(String args[]) {

            Integer[] array1 = { 2, 3, 1 };
            Integer[] array2 = Arrays.copyOf(array1, array1.length);
            Arrays.parallelSort(array2);

            Optional<String> a =
                  Optional.ofNullable(Question_46.class.getSimpleName());
            int b = 0;

            for (char x : a.get().toCharArray()) {

                  Arrays.stream(array1).forEach(y -> {
                      if  (String.valueOf(x).equals(String.valueOf(y)))  {
                              b++;
                              System.out.println(b);
                              return;
                      }
                  });

            }

      }
}
```

(A) output nothing
(B) output 1
(C) compile error
(D) runtime exception

Answer : (C) compile error

What this program does is below. (if compile error doesn't occur).

1. create two arrays and sort second one
2. loop charArray from String “Question_46”
3. check if array1 includes letter in it
4. if condition at line 15 never returns true, so program ends without output anything

Notes :

(1) if condition at line 15

At line 15, it checks that elements in array1 includes any letter of String “Question_46”.

The elements are [1, 2, 3]. So, this condition doesn't return true even after evaluation of all elements.

(2) change value in lambda

At line 14 to 20, it is lambda expression. In lambda expression, you can't change value of variable in it. It should be final.

If code at line 14 to 20 is below, the answer is (A).

```
for (Integer element : array1) {
      if (String.valueOf(x).equals(String.valueOf(element))) {
            b++;
            System.out.println(b);
            return;
      }
}
```

(3) at line 4

If you change element 4 to 2 of array1 at line 4, the answer is (B).

Question 47

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_47 {

      public Question_47() {
            Random a = new Random();
            String b = "a";
            Integer c = null;
            int abc = 0;

            try {
                  c = a.nextInt(Integer.parseInt(b));
            } catch (Exception e) {
                  abc += 1;
                  return;
            } finally {
                  Optional<Integer> d = Optional.ofNullable(c);
                  System.out.print(d.orElse(abc));
            }
      }

      public static void main(String args[]) {
            Question_47 a = new Question_47();
      }

}
```

(A) output nothing
(B) output 1
(C) compile error
(D) runtime exception

Answer : (B) output 1

What this program does is below.

1. create instance of Question_47
2. try to convert String “a” to int
3. throws NumberFormatException
4. output 1

Notes :

(1) constructor

At line 3 to 18 is constructor of class Question_47. This will be excuted when instance of class Question_47 is created .

So, this will be executed at line 21.

(2) Processing order of return statement and finally block

At line 13, there is return statement. However, before returning, finally block will be executed first.

After execute finally block, return statement will be executed.

(3) method orElse()

This method in Optional class is for nullable value.

If you use if statement, you have to write code like below instead of code at line 16.

```
if (object != null) {
      System.out.print(d);
} else {
      System.out.print(abc);
}
```

Question 48

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_48 {

      private static Predicate<Integer> predicate = Predicate.isEqual(1);

      public static void main(String args[]) {
            List<Integer> list1 = Arrays.asList(1, 2, 3, 4, 5);
            List<Integer> list2 = Arrays.asList(1, 2);
            List<Integer> list3 = Arrays.asList(3, 4, 5);
            List<List<Integer>> list4 = Arrays.asList(list2, list3);
            List<Integer> list5 = null;

            Optional<List<Integer>> a = Optional.ofNullable(list5);
            try {
                  a.orElseThrow(NullPointerException::new);
            } catch (Exception e) {
                  list5 = list4.stream()
                              .flatMap(b -> b.stream())
                              .sorted(Comparator.reverseOrder())
                              .collect(Collectors.toList());

                  System.out.print(predicate.test(list1.get(0))
                        && predicate.test(list5.get(list1.size() - 1)));
            }
      }
}
```

(A) output true
(B) output false
(C) compile error
(D) runtime exception

Answer : (A) output true

What this program does is below.

1. create list1 with elements [1, 2, 3, 4, 5]
2. create list2 with elements [1, 2]
3. create list3 with elements [3, 4, 5]
4. create list4 with elements [list2, list3]
5. create list5 as null and wrap it with method ofNullable()
6. throw NullPointerException
7. assign elements to list5 with reversed order [5, 4, 3, 2, 1]
8. output the result of comparing if 1 equals to the first element of list1 and last element of list5

Notes :

(1) method orElseThrow()

This method in Optional class is for nullable value. At line 12, list5 is null at this point, so line 14 throws NullPointerException.

This method throws NullPointerException if object a is null.

And, you can rewrite code at line 14 without using method reference like below, too.

```
a.orElseThrow(() -> new NullPointerException());
```

(2) method flatMap()

This method changes multiple collections into one stream. It makes it easy to concat list.

(3) Predicate

What code at line 3, 21 and 22 does simple. It compares 1 and first element of list1 and last element of list5

This is way to use method isEqual() in Predicate Interface.

Question 49

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_49 {

    private static final String str;
    private static final String[] array;

    static {
        str = "a,b,c";
        array = str.split(",");
    }

    public static void main(String args[]) {
        Arrays.stream(array).forEach(e -> {
            switch (e) {
            case "a":
                System.out.print(1);
            case "b":
                System.out.print(2);
            case "c":
                System.out.print(3);
            default:
                System.out.print(4);
            }
        });
    }

}
```

(A) output 123
(B) output 123423434
(C) output 123412341234
(D) compile error or runtime exception

Answer : (A) output 123423434

What this program does is below.

1. initialize two fields str and array
2. loop the array and output with switch statement

Notes :

(1) How this switch statement is executed

At line 13 to 22, there is no break; statement. So, the processing order is like below.

first loop. line 15, line 17, line 19, line 21
second loop. line 17, line 19, line 21
third loop. line 19, line 21

This is why answer is (B). If code at line 13 to 22 is below, the answer is (A).

```
switch (e) {
case "a":
      System.out.print(1);
      break;
case "b":
      System.out.print(2);
      break;
case "c":
      System.out.print(3);
      break;
default:
      System.out.print(4);
}
```

In this case, default case is not executed, because all cases match elements of the array.

The default case is executed if there is no matching.

Question 50

Select the one statement that describes the result when this program is compiled and run.

```
public class Question_50 {

      private static Predicate<Integer> a = new Predicate<Integer>() {
            public boolean test(Integer b) {
                  return b > 0;
            }
      };

      public static void main(String args[]) {
            try {
                  String[] array = { " 12 3 ", "123 " };

                  for (String str : array) {
                        str = str.replace(" ", "");
                  }

                  System.out.print(a.test(Integer.parseInt(array[0])));
            } catch (Exception e) {
                  System.exit(1);
            } finally {
                  System.out.print("done");
            }
      }

}
```

(A) output nothing
(B) output truedone
(C) output done
(D) compile error or runtime exception

Answer : (A) output nothing

What this program does is below.

1. create array with elements [" 12 3 ", "123 "]
2. loop it and replace space to empty string
3. terminate application

Notes :

(1) standard for loop, enhanced for loop, forEach of StreamAPI

There are three ways to loop array, but there is only one way that affects the root array like below.

1. standard for loop (affects root array)
2. enhanced for loop (doesn’t affect root array)
3. forEach of StreamAPI (doesn’t affect root array)

This is why the answer is (A).

If you use forEach of StreamAPI at line 14 to 16, the answer is also (A).

If you use standard for loop at line 14 to 16, the answer is (B), because the loop replaces all space to empty string.

NumberFormatException from method Integer.parseInt() doesn’t occur if all spaces are replaced to empty string.

(2) way to stop application

At line 19, method System.exit() is used to stop application.

In this case, finally block is not executed.

If you write “return;” at line 19 instead of “System.exit(1);”, the answer is (C).

Copyright © MilkTea

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the publisher except for the use of brief quotations in a book review.